The Star Solution

Komi Agbo, Dalton Burke, Nick Mako, James Vance

December 9, 2019

Abstract

Sam's Hauling, Inc. provides waste disposal containers throughout the Denver metro area. In order to increase efficiency they are seeking a way to create delivery schedules for their drivers, subject to a zoo of constraints. Our solution averts standard Traveling Salesman Problem style approach for something tailored specifically for the problem. The result, written in Python 3.7[6], is a fast, nearly optimal solution which takes advantage of how central Landfills and Storage Yards are to the problem.

Keywords - Optimization, Constrained VRP, Assignment Problem

1 Introduction

Sam's Hauling, Inc. provides roll-off dumpsters (colloquially, *cans*) of various sizes to homeowners, contractors, realtors and property managers throughout the Metro Denver area. Each job site (delivery, pickup, or switch) has some constraints; they need a particular can size; they often need it at a certain time (AM, PM, or unrestricted); sometimes they need a certain sized truck to do it. They have a limited number of trucks with which to do these pickups and deliveries and in addition some customers set time windows for said pickups and deliveries. There are multiple depots where they store the cans, which start the day with a certain number of cans in stock.

At first blush, the problem seems to be some nightmarish amalgamation of complicated constraints and NP-Hard[1] problems, however, its saving grace is that trucks can carry only one container at a time, and must return to a landfill or a storage yard between jobs frequently.

Currently, the pickup and delivery of these cans is scheduled by a individual using only Microsoft Excel. This method for solving it is very time intensive and is less likely to be the most efficient solution. It is quite likely then, that the implementation of some heuristic or metaheuristic could increase the efficiency of this process by a significant margin.

2 Background

The Sam's Hauling Problem initially seems very similar to the Traveling Salesman Problem[4] which asks the question, "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" If the TSP had multiple salesmen, (as in the Vehicle Routing Problem[5], VRP), we'd need to figure out a route for each driver so that they visit every job, and return to the home base at the end of the day.

The parallels are obvious, however, the biggest problem with the TSP and the VRP is that they are both NP-Hard, which makes them computationally infeasible to find optimal solutions for. To solve these problems quickly requires heuristic methods, which can still be computationally expensive, and will almost certainly not give an optimal solution.

Our approach doesn't depend very much on known algorithms, outside of one, the Hungarian Algorithm[3], which is a polynomial time algorithm ($O(n^3)$) that generates an optimal solution to an assignment problem[3]. In our application, we want to pair up deliveries and pickups so that the total route cost (over all drivers) is minimized.

3 Methods

Instead of thinking of the problem as a Vehicle Routing Problem, where the order of deliveries directly effects the cost of the route. Here, we know that for

the most part, order doesn't matter. This is because of how frequently storage yards and landfills are visited.

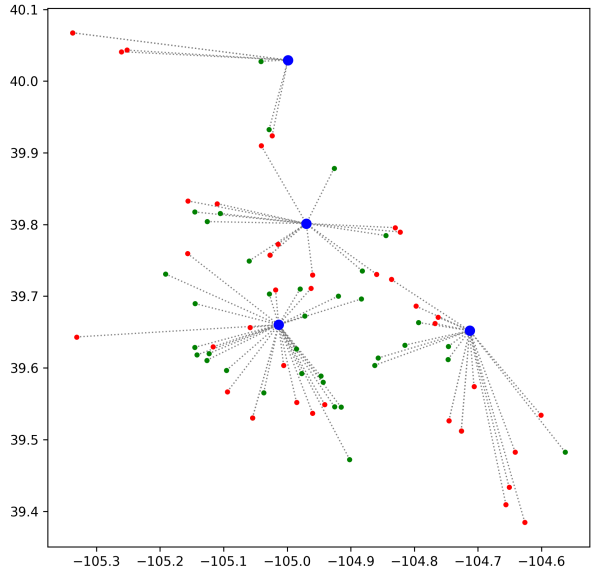
Note: since for each storage yard there is a nearby landfill, we will refer exclusively to the storage yard.

3.1 Stars

If customers only ever wanted container switches, the problem would be much simpler. Where should the truck come from that's going to deliver the new container? The nearest storage yard. And where should the truck go after picking up the full container? The nearest landfill.

These choices fulfill a kind of local optimality, however, they may not lead to a solution which is possible. Remember, we want to have drivers begin and end their day at the home base, so how can a driver service a job in Erie if they start in Denver? Another question, what about the inventory of that storage yard, or if a specific truck needs to be the one to do that job? These extra factors will be fixed after the fact, since most of the day is spent working in these locally optimal situations.

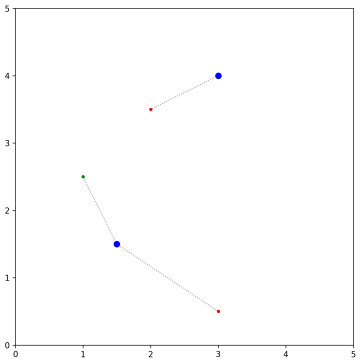
Filling the route with this data, we end up with several *stars*, each storage yard being a center, and the points surrounding it being the job sites. To service each star, you can take the jobs in any order, or with any driver, the resulting cost will be the same. This will give us a lot of leniency when trying to satisfy the constraints. The only task remaining is to sew them together into routes for the drivers.



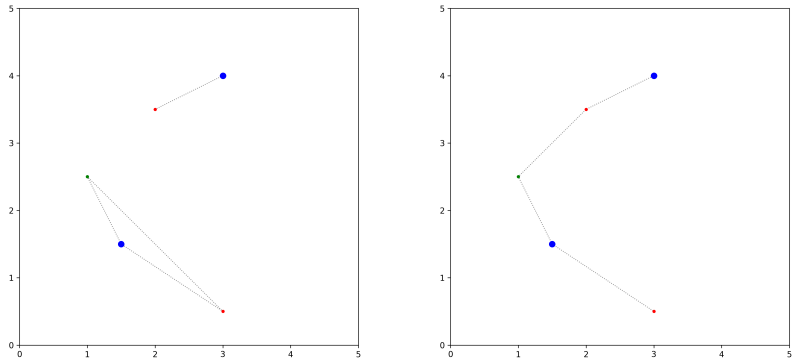
Blue dots are storage yards, green dots are pickups, red dots are deliveries. Figures made in matplotlib[2]

3.2 Triangles

Since the problem doesn't actually consist solely of switches it is necessary to expand the star solution to include pickups and drop-offs. This provides a route of storage yard, drop-off, pick-up, storage yard for a driver, creating a triangle. These triangle routes allow for further optimization as long as the optimal triangles are chosen.



Small example stars



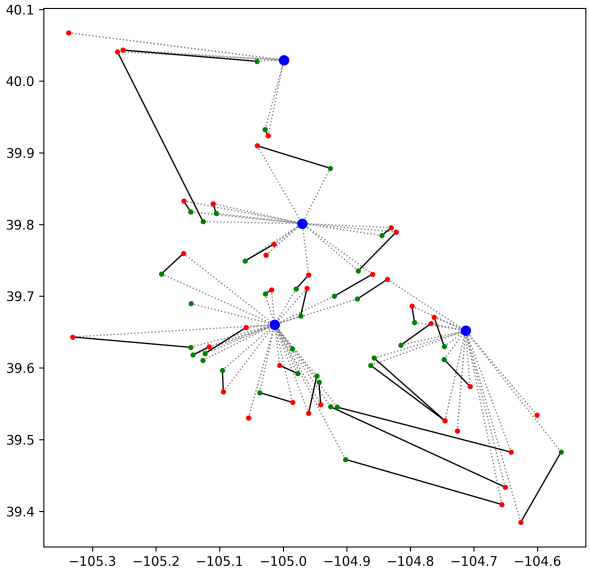
On the left we have already done better than the star example above, however, the configuration on the right requires less driving, and moves a driver from one storage yard's zone to another.

Some triangles are better than others, so what we want is the best *set* of triangles, those which minimize the total amount of driving over the entire map. This type of problem has been studied before, it is known as the Assignment Problem[3], and the people who studied it made an extremely efficient way to solve it optimally (no approximations!), known today as the Hungarian Algorithm[3]. We make use of an implementation of that algorithm[7] to calculate that optimal triangle setup, keeping in mind compatibility between pickups and deliveries.

In order to use the algorithm, we need to associate a "weight" to every pair. The algorithm generates the best pairs based on that weight. To calculate these

weights, we take the difference between what the pair would cost individually, as a star, and what they would cost together as a triangle. Minimizing this maximizes the "benefit" of the triangles over the stars, which is exactly what we want.

We allow for pairs to cross between storage yard zones, that is, the delivery has one closest storage yard and the pickup can have a different closest storage yard. In a matching where this is allowed, the result is almost always better than restricting pairs to the same zone. One unfortunate consequence is that the name "triangle" becomes less intuitive.



Deliveries and Pickups matched from real data.

3.3 Route Building

Now that we have matched up deliveries and pickups optimally, we need to allocate the jobs to the drivers in a reasonable way. For this, there are three considerations, how do we move drivers between storage yard zones? (transitions), how do we satisfy truck and time of day constraints? And how do we manage the supply of cans among the storage yards?

3.3.1 Transitions

In order to determine each transition, we calculate $\min(RN_i - RT_i)$ for $i = 0$ to $i = n$ where RN is the route cost normally, RT is the cost of doing the transition, and n is the number of job pairs that have not been assigned to a driver. In the case where pairs are constrained $\min(RN_i - RT_i)$ will always be positive since one of the two endpoints is no longer to the closest storage yard.

However in the case where pairs are non-constrained and can be made between zones the cost of doing a job pair normally can be greater than the cost of doing a transition since the pair can have different storage yards that are closest.

3.3.2 Non Transition Routes

Non-transition routes are first assigned to fulfill constraints, second to match the size of the can the driver just picked to the size of can to be delivered next and are assigned in descending order based on their distance to the main hub of the company.

3.3.3 Truck and Time Constraints

Two constraints that need to be considered are the time constraint and the truck size constraint. Customers may request a delivery in specifically the AM or PM hours of the day. In order to deal with this constraint, routes identified as such are assigned with priority when a driver's schedule is within the time period. In addition, some trucks are restricted as to where they can deliver cans. This constraint is resolved by identifying routes that require a specific truck and assigning the first driver that truck with a special schedule that services all routes with the constraint and then performs as a regular driver within the zone they end in. If more Truck constraints exist or the first driver did not complete all of the constraints for the size he was assigned another driver will be assigned in the same manner until this is resolved

3.3.4 Can Supply Management

A final issue considered is the supply of cans at a given depot. There are two types of can deficits that must be dealt with. First are hard can container deficits which occurs when there are more cans that need to leave the storage location than will be returned plus the initial stock on a given day. These can be resolved in two stages. The first stage is for a deficit of size up to the number of drivers that will start their day driving to the zone with a deficit. In this stage the calculation for transitioning will be altered to only consider routes with a pickup of the appropriate size can outside the zone with the deficit. In the second stage where the problem cannot be resolved in the initial stage the first driver assigned to the area will be assigned special transitions that both originate and end in the deficit zone, but which leave in order to go to a pickup of the size of the can with a deficit. A soft can deficit, which means that the number of returns plus the initial stock of cans at a depot is at least as large as the number of cans of a particular size that need to be delivered. This can still be an issue if too many deliveries happen before the pickups for a particular size of can. Even where returns plus the initial stock are at least as large as the number of cans of a particular size, a problem may occur if too many deliveries happen before the pickups for a particular size of can. To resolve this the draft schedule is simulated through the day and where conflict arises a swap will

be made to fix the problem with priority given to swapping within the same driver's schedule. Due to how our solution works these swaps will not affect the efficiency. When considering whether an issue exists other driver's schedules will be considered to be at a time less than the current simulated time by some amount of minutes L to account for potential differences due to delays.

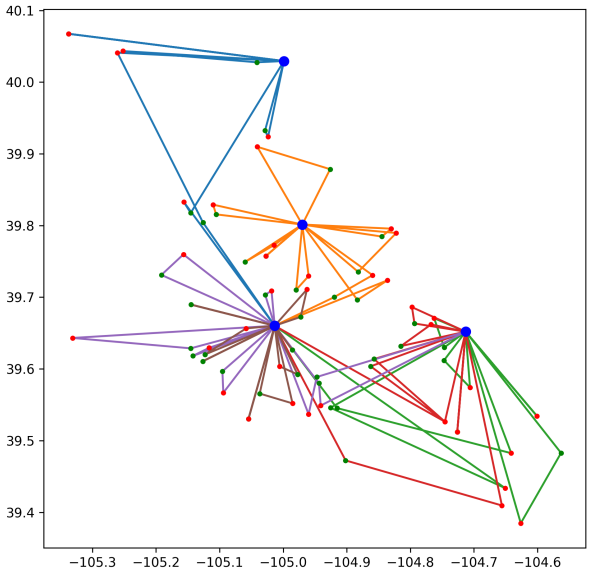
3.4 Recap

The algorithm for assigning the full schedule are as follows:

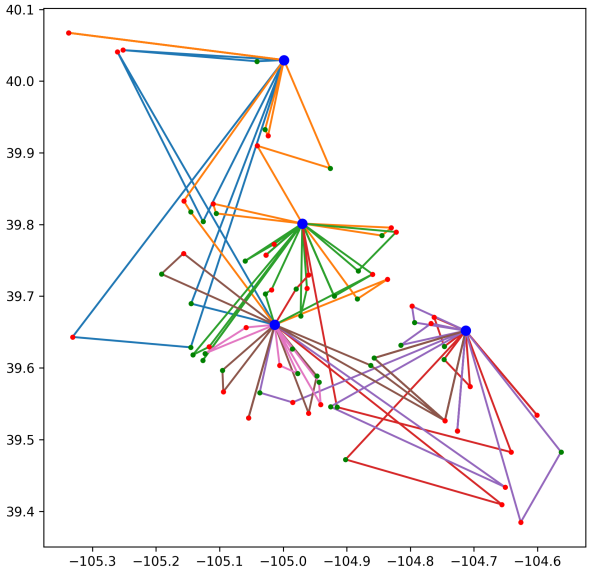
1. Receive data for the day and process it into the program.
2. Create pairs with Hungarian algorithm.
3. Assign driver to resolve truck size constraint.
4. Transition to the farthest zone that requires the truck resolving a hard can deficit if applicable.
5. Service all routes within the zone that require the truck and return to step 2 if more applicable routes exist in other zones.
6. Finish route as per a normal driver and return to 1 if other truck size constraints exist.
7. Assign driver to farthest zone with routes.
8. Transition to the zone.
9. If applicable resolve hard can deficits.
10. Assign remaining routes by prioritizing AM/PM constraints, then routes where the delivery can size is the same as the pickup from the prior pair of routes, and finally distance to the main hub.
11. If there are no more routes in the zone transition to the next zone and return to step 8.
12. If the driver schedule is full including the transition back to the main hub transition back to the main hub and return to step 7 with a new driver.
13. Iterate through the day to resolve any soft can deficits making swaps as needed.
14. Output schedules.

4 Results

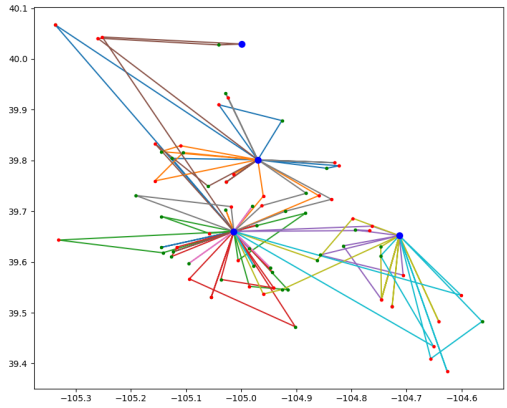
From an old schedule provided by the client, we ran our algorithm to see how it stacked up to the clients solution. In the following figures, each color is a driver's route.



Routes generated by the algorithm. Total time spent driving: 1576 minutes.



Routes generated by the algorithm, including severe can restrictions. Total time spent driving: 1781 minutes.



Route provided by client. Total time spent driving: 1945 minutes.

5 Discussion

Here are some comparisons:

Name	Minutes	% TP	% Sam's
Teleportation	1534	0.0	21.1
No Can Constraints	1576	2.7	18.9
Can Constraints	1781	16.1	8.4
Sam's	1945	26.8	0.0

Here, "Teleportation" refers to the route if drivers could teleport between storage yard zones, that is, we don't need any transitions (just the triangles by themselves). The third column is the percent time lost compared to the teleportation route, and the fourth column is the percent time saved over the schedule given by the client.

One thing to note about this output is that requiring transitions only added 2.7% to the time of the solution. To us, that indicates that our pair selection did quite well, as we could only potentially save that much more.

The program runs very quickly, and it outperforms the client's attempt by 8.4%. We would have liked to run more data sets to get a more comprehensive report, however, our access to data is limited.

6 Conclusions/Future Work

There are two things that would make this program better. One, is better data. If given actual driving times, the output would do a much better job of reflecting

real world conditions. As of now, we actually can't even test how well it's doing.

The other thing that would make this better is a better pair choice reflecting the transitions that will inevitably need to be taken. Right now, pairs are chosen as if drivers can teleport between storage yards. While a vast majority of a drivers time is *not* taking transitions, it still seems like we are losing a bit here. In order to truely take them into account however, it seems like a much more painful implementation than what is presented.

In the end, having to take the transitions (as opposed to teleporting between storage yards) cost the route about 42 minutes, out of 1576, which is around a 2.7% difference. An improvement could only save us about that much. The obvious next step is to find a bound for the difference between teleportation and our output, however, due to time constraints, we leave it unanswered for now.

References

[1] Johnson Garey. *Computers and Intractability.* W. H. Freeman, 1979.

[2] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[3] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[4] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231 – 247, 1992.

[5] Suresh Nanda Kumar and Ramasamy Panneerselvam. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 04, 01 2012.

[6] Python Core Team. *Python: A dynamic, open source programming language.* Python Software Foundation, 2019. Python version 3.7.

[7] Unknown. *munkres — Munkres implementation for Python.* A python package.